

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**FAULT TOLERANCE IN THE
SERVER AND AGENT BASED
NETWORK MANAGEMENT (SAAM) SYSTEM**

by

Troy Wright

September 2001

Thesis Advisor:
Second Reader:

Geoffrey Xie
Bert Lundy

Approved for public release; distribution is unlimited

Report Documentation Page		
Report Date 30 Sep 2001	Report Type N/A	Dates Covered (from... to) -
Title and Subtitle Fault Tolerance in the Server and Agent Based Active Network Management (SAAM) System	Contract Number	
	Grant Number	
	Program Element Number	
Author(s) Wright, Troy V.	Project Number	
	Task Number	
	Work Unit Number	
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey Ca. 93943-5138	Performing Organization Report Number	
Sponsoring/Monitoring Agency Name(s) and Address(es)	Sponsor/Monitor's Acronym(s)	
	Sponsor/Monitor's Report Number(s)	
Distribution/Availability Statement Approved for public release, distribution unlimited		
Supplementary Notes		
Abstract		
Subject Terms		
Report Classification unclassified	Classification of this page unclassified	
Classification of Abstract unclassified	Limitation of Abstract UU	
Number of Pages 59		

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2001	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Fault Tolerance in the Server and Agent Based Active Network Management (SAAM) System			5. FUNDING NUMBERS	
6. AUTHOR(S) Wright, Troy V.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) Interconnected networks of computers are becoming increasingly important. It is the Internet that has spurred the most recent growth in global computer networks. The limitations of the Internet can be blamed on many factors but when determining solutions to these shortcomings the focus has been on replacing the current Internet Protocol version 4 (IPv4) with the new Internet Protocol version 6 (IPv6). Much work has been done and much more work remains to be done in transitioning to and reaping the benefits of this "Next Generation Internet." The Server and Agent Based Active Network Management (SAAM) project is one of many "Next Generation Internet" projects that intend to implement and exploit the enhanced capabilities of IPv6 to overcome the limitations of the current Internet. The focus of the SAAM project is guaranteed quality of service (QoS). This thesis addresses fault tolerance in a SAAM region with regards to router and link failures. A hybrid link restoration (rerouting) scheme is proposed, in which central knowledge (at the SAAM server) of the network topology is used to develop alternate paths while path switching is done at a local (router) level.				
14. SUBJECT TERMS Fault Tolerance, Local Rerouting, Next Generation Internet, Guaranteed Quality of Service			15. NUMBER OF PAGES	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**FAULT TOLERANCE IN THE SERVER AND AGENT BASED
NETWORK MANAGEMENT (SAAM) SYSTEM**

Troy V. Wright
Captain, United States Marines Corps
B.A., University of Utah, 1992

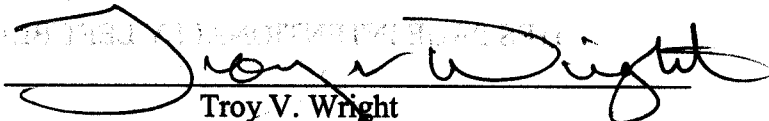
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE


from the


**NAVAL POSTGRADUATE SCHOOL
September 2000**


Author:


Troy V. Wright

Approved by:


Geoffrey Xie, Thesis Advisor


Bert Lundy, Second Reader


Chris Eagle, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Interconnected networks of computers are becoming increasingly important. It is the Internet that has spurred the most recent growth in global computer networks. The limitations of the Internet can be blamed on many factors but when determining solutions to these shortcomings the focus has been on replacing the current Internet Protocol version 4 (IPv4) with the new Internet Protocol version 6 (IPv6). Much work has been done and much more work remains to be done in transitioning to and reaping the benefits of this “Next Generation Internet.” The Server and Agent Based Active Network Management (SAAM) project is one of many “Next Generation Internet” projects that intend to implement and exploit the enhanced capabilities of IPv6 to overcome the limitations of the current Internet. The focus of the SAAM project is guaranteed quality of service (QoS). This thesis addresses fault tolerance in a SAAM region with regards to router and link failures. A hybrid link restoration (rerouting) scheme is proposed, in which central knowledge (at the SAAM server) of the network topology is used to develop alternate paths while path switching is done at a local (router) level.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	1
B.	PROBLEM STATEMENT AND APPROACH.....	1
C.	SCOPE	2
D.	THESIS ORGANIZATION.....	2
II.	NETWORK FAULT TOLERANCE AND SAAM BACKGROUND.....	3
A.	GOAL OF SAAM	3
B.	SAAM ARCHITECTURE	3
C.	LINK RESTORATION.....	5
1.	Types of network failures.....	5
2.	Current rerouting solutions	5
3.	Next generation rerouting solutions	6
III.	FAILURE DETECTION.....	9
A.	FAILURE DETECTION STRATEGY.....	9
B.	SAAM AUTOCONFIGURATION CYCLE	9
1.	Pseudo-code for proposed SAAM failure detection scheme	10
2.	Sample SAAM auto-configuration cycle.....	11
C.	INTERFACE SILENT MESSAGE.....	13
D.	FAILURE DETECTION LIMITATION	15
E.	CLASSES MODIFIED TO SUPPORT FAILURE DETECTION	15
1.	Interface.....	15
a)	<i>public boolean trafficReceived().....</i>	<i>15</i>
2.	LinkStateMonitor	15
a)	<i>private synchronized void generateInterfaceSA().....</i>	<i>15</i>
3.	InterfaceSA.....	15
a)	<i>public InterfaceSA(IPv6Address ipNum, int messageIndex).....</i>	<i>15</i>
4.	BasePIB.....	15
a)	<i>public void processLSA (LinkStateAdvertisement LSA)</i>	<i>15</i>
IV.	REROUTING.....	17
A.	REROUTING STRATEGY.....	17
1.	Alternate Path Development and Deployment.....	17
a.	<i>Pseudo-code for Alternate Path Development and Deployment.....</i>	<i>17</i>
2.	Storing Primary and Alternate Routes.....	19
3.	Local restoration	20
B.	CLASSES MODIFIED TO SUPPORT REROUTING.....	21
1.	FlowRoutingTable.....	21
a)	<i>public synchronized void add (FlowRoutingTableEntry entry).....</i>	<i>21</i>
b)	<i>public Object get(Object o)</i>	<i>21</i>
c)	<i>public void silentInterface(Interface badInterface).....</i>	<i>21</i>

2.	FlowRoutingTableEntry.....	22
a)	<i>public int getGoodness()</i>	22
3.	BasePIB.....	22
a)	<i>private Path findAltPath(Integer sourceNode, Integer destinationNode, Integer previousNode, Integer nextNode, IPv6Address deadInterface)</i>	22
b)	<i>protected void createAlternateTree(Path primaryPath)</i>	22
c)	<i>protected int admissionControl_IS(FlowRequest flowRequest)</i>	22
IV.	TESTING.....	23
A.	ALTERNATE PATH(S)/TREE TESTING.....	23
1.	No alternate paths topology	23
2.	Avoid next interface on last hop topology.....	24
3.	Avoid infinite loop topology	26
4.	Avoid next node topology	27
5.	Avoid next interface topology	29
B.	TRAFFIC REROUTING TESTING	30
V.	CONCLUSIONS AND RECOMMENDATIONS.....	33
A.	SYNOPSIS AND CONCLUSION	33
B.	FLOW ROUTING TABLE REDESIGN.....	33
C.	FAILURE DETECTION.....	33
D.	TEST METHODOLOGY	34
E.	AREAS FOR FURTHER INVESTIGATION AND STUDY	34
APPENDIX A.	FLOW ROUTING TABLE CODE CHANGES	37
APPENDIX B.	BASEPIB CODE CHANGES.....	39
	LIST OF REFERENCES.....	43
	DISTRIBUTION LIST	45

LIST OF FIGURES

Figure 2.1	Sample SAAM Network.	4
Figure 3.1	Auto-configuration Legend.....	11
Figure 3.2	Auto-configuration Phase 0.	12
Figure 3.3	Auto-configuration Phase 1.	12
Figure 3.4	Auto-configuration Phase 2.	12
Figure 3.5	Auto-configuration Phase 3.	13
Figure 3.6	Auto-configuration Phase 4.	13
Figure 3.7	Interface Silent Message Format	14
Figure 5.1	No alternate path topology.....	23
Figure 5.2	Avoid next interface topology.	25
Figure 5.3	No infinite loops topology.	26
Figure 5.4	Avoid next node topology.....	28
Figure 5.5	Avoid next interface topology.	30

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 4.1	Example of Old Flow Routing Table.....	20
Table 4.2	Example of New Flow Routing Table.	20
Table 4.3	Flow Routing Table after Switching to Backup Path(s).....	21
Table 5.1	No Alternate Path Routing Tables.	24
Table 5.2	Avoid Next Interface Routing Tables.	25
Table 5.3	No Infinite Loops Flow Routing Tables.	27
Table 5.4	Avoid Next Node Flow Routing Tables.	29
Table 5.5	Avoid Next Interface Flow Routing Tables.....	30
Table 5.6	Router “B” Interface Status	31
Table 5.7	Router “B” Link State Monitor.....	31
Table 5.8	Router “A” Link State Monitor.....	31
Table 5.9	Router “A” Flow Routing Table	32
Table 5.10	Router “C” Flow Sink.....	32

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to thank Professor Xie for his patience and assistance in this endeavor. His guidance allowed me to explore and learn while maintaining a steady course towards the destination.

I. INTRODUCTION

A. MOTIVATION

Interconnected networks of computers are becoming increasingly important. Most people interact or are affected by computer networks on a daily basis. Computer networks have been around for quite some time but it is the Internet that has spurred the recent rapid growth in computer networks.

The Internet is constantly evolving and providing new and innovative ways to improve the way we conduct business and our lives in general. Due to the numerous ways that people have found to exploit the capabilities of the Internet the network has become overtaxed in a number of ways. Because of the limitations of the current Internet, specifically Internet Protocol version 4 (IPv4), a new protocol Internet Protocol version 6 (IPv6) has been designed and research into the “Next Generation Internet” has been embarked upon.

The Server and Agent Based Active Network Management (SAAM) project is one of many “Next Generation Internet” projects, which intends to implement and exploit the enhanced capabilities of IPv6 to overcome the shortcoming of the current Internet.

B. PROBLEM STATEMENT AND APPROACH

The SAAM project was launched in 1998 as the thesis project for two computer science students at the Naval Postgraduate School. Since its humble beginnings it has served as the thesis vehicle for more than 20 students.

Currently the SAAM project could be classified as being in the intermediate phase of development. The basic network routing functions have been designed and implemented and a graphical user interface has been overlaid to provide a more efficient method of monitoring the working SAAM network and speed development efforts.

One of the pillars of the SAAM project is guaranteed quality of service (QoS) for certain classes of flows admitted into a SAAM network region. When developing a plan to guarantee QoS to network flows the possibility of network failures must be taken into consideration. Dealing with network failures is the focus of this thesis.

The approach to this thesis will be to identify possible points of failure within the SAAM network. Continue by determining the probability of failure at those points identified. Once the probability of failure has been determined consider which points have a failure probability that is significant enough to warrant proactive or reactive measures by the SAAM network to be developed for them. Finally after identifying the potential points of failure, which have significant fault probabilities, develop fault tolerant measures to compensate in the event that one of those failures does occur.

C. SCOPE

The efforts of previous participants in the SAAM project have laid the foundation. This thesis will add functionality to already existing objects such as the Link State Monitor in achieving the goal of fault tolerance. Although the majority of the software coding anticipated in this thesis will be modifications to current modules, sound software development principles will be adhered to at all times, in particular a focus will be maintained on developing modular code whenever possible.

The scope of this thesis will be limited to three primary tasks: alternate path (tree) development, fault detection, and rerouting affected flows upon fault detection.

D. THESIS ORGANIZATION

- Chapter I: Introduction. A brief description of the problem addressed by this thesis.
- Chapter II: Network Fault Tolerance and SAAM Background.
- Chapter III: Failure Detection & Path Switching.
- Chapter IV: Creating Alternate Path(s)/Tree(s).
- Chapter V: Testing.
- Chapter VI: Conclusions and Recommendations.

II. NETWORK FAULT TOLERANCE AND SAAM BACKGROUND

A. GOAL OF SAAM

The primary purpose of the Server and Agent Based Network Management project is to provide quality of service (Quality of Service) guarantees in the next generation Internet. The SAAM project intends to take advantage of the increasingly fast physical communication media by streamlining router functionalities and transferring most processing intensive tasks from all routers to just a few SAAM Servers.

One of the key considerations during the development of SAAM is compatibility with the current Internet. The extremely slow adoption of IPv6 is evidence that most businesses have a significant amount of capital invested in IPv4 hardware and the benefits of IPv6 do not currently justify the cost of purchasing new hardware. SAAM must be compatible with both IPv4 and IPv6 while providing affordable QoS guarantees.

B. SAAM ARCHITECTURE

A SAAM region is a network of SAAM routers and servers. One server, the primary server, monitors and controls all the activity within a SAAM region.

Server and Agent based Active network Management

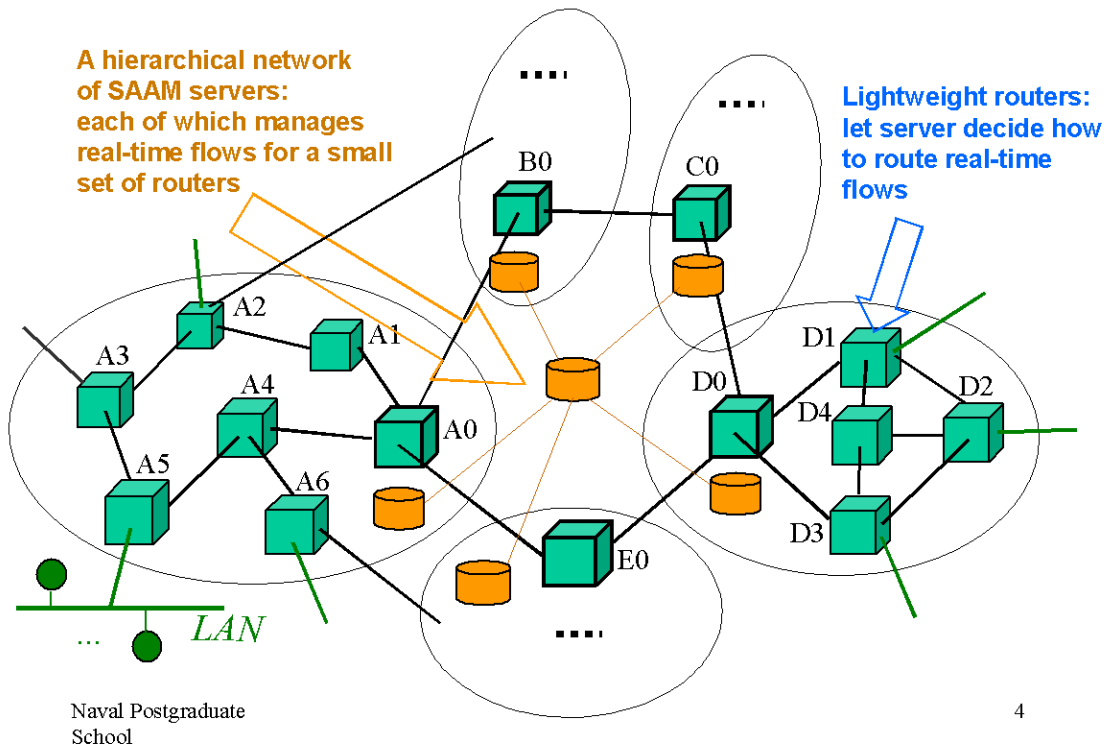


Figure 2.1 Sample SAAM Network.

One of the distinguishing features of the SAAM server is its total knowledge of the network topology and status. The server maintains a Path Information Base (PIB), which is a table of all known paths within a SAAM region. The PIB is populated as routers are added to the region. The rerouting strategy of this thesis relies on the PIB to determine alternate paths and in doing so does not incur any additional overhead in route discovery because only paths already within the PIB are utilized.

If there is more than one server within a SAAM region the other servers are backup servers and can take over in the event of primary server failure or malicious attacks on the primary server. Efraim Kati's thesis, "Fault Tolerant Approach for Deploying Server Agent Based Active Network Management (SAAM) Server in Windows NT Environment to Provide Uninterrupted Services to Routers in Case of Server Failure(s)", addresses the issue of fault tolerance in the event of SAAM server failures(s).

A SAAM region is self-configuring. Configuration and control traffic is constantly passed between the routers and the server. This traffic keeps the server abreast of current network load and health. Of critical importance to this thesis is the Link State Advertisement (LSA) message. The LSA is a report to the server on the status of a router interface. Every time there is an auto-configuration cycle, each router sends one LSA for each of its interfaces to the server.

C. LINK RESTORATION

1. Types of network failures

There are primarily three network components that may fail, and these three can be placed in two categories. The first component that may fail is the wire, optical fiber, or whatever type of media carries the signal between interface cards. The second component that may possibly fail is the interface card, which connects the wire and the router. The final component that may fail is the router itself. The interface card and the wire are combined into one entity called a link. The router itself is called a node. An entire network topology is a set of nodes connected by a set of links. Of the two types of failures, the link failure is far less serious than a node failure. When a node fails, in effect, it causes all its connected links to fail. If there is an interface failure the router which hosts the interface will discover the failure and will broadcast a revised routing table to the routers that are attached to its good links. If a link failure or node failure occurs all the routers attached to the failed link/router will detect the failure, will revise their routing tables accordingly, and will rebroadcast their routing tables. These failures are detected because the routers exchange routing information on a periodic basis. If a neighboring router fails to respond within a set amount of time its neighbors will consider that link dead. At this point any arriving traffic will continue through to its destination because new routing tables have been computed, with the failed router/link being taken out of the network topology.

2. Current rerouting solutions

The current solution to rerouting is handled by the same mechanism that creates the primary routes for network traffic. When a router is brought online it broadcasts its existence to its neighboring routers. Its neighbors receive this broadcast message and make an entry in their routing tables, indicating that they can now reach the new router

with a cost of 1 hop. The hop metric is the most commonly used cost metric in routing tables. The routers that receive the broadcast message reply to the new router with their own routing tables, which the new router records for future use. An administrator can enter certain static routes and default routes, which do not or should not change very often, into the router. Now that the new router has exchanged routing information with all its neighbors it can make a next hop routing decision based on the Internet Protocol (IP) number and the cost metric, if more than one possible route presents itself. The process mentioned above is implemented with standard protocols such as the Routing Information Protocol (RIP) and Open Shortest Path First (OSPF).

What are the drawbacks of this solution? The main drawback is that any TCP (transport layer) sessions that existed on the link/node when the failure occurred will be lost. The user or application will have to renegotiate a TCP session after the failure has been discovered and the new routes have been determined. Also, it should be noted that the advertisement of failures in this type of network could take a very long time to propagate through the network, causing significant delay in reestablishing TCP sessions.

Are there any ways to improve on the current solution without replacing it entirely? One of the simplest quick fixes to a network failure is to instruct the router to send packets destined for the failed link to the default link, or any available link, in hopes that the packets will manage to get to their destination. This is a quick and simple fix but it lacks reliability.

The main problems in the current solution that should be addressed by any next generation solution are the timely discovery of network failures and the expeditious rerouting of the affected traffic. As mentioned before the big push for QoS has driven the need for better network rerouting. The goal of this next generation rerouting system is to recover from any network failures so quickly that the TCP session is not dropped and that a minimal number of packets are lost during the transition.

3. Next generation rerouting solutions

One of the most popular concepts for next generation rerouting is an entity that monitors the health of the entire network. This network monitor could keep an eye on the current status of all links/nodes using reserved channels on existing links or auxiliary

links if deemed necessary. Instead of reporting their status to their neighbors, the nodes in this type of network would report their status to the network monitor. The network monitor could then create a picture of the network and send only pertinent routing information to each node in the network.

The detection of failures in this type of network would occur much more rapidly. When a failure was detected the detecting entity need only notify the network monitor of the failure. The affected routes would be quickly updated by the monitor and new routing information sent only to those destinations which required it.

If QoS is a critical issue, this network monitor could provide primary label switched paths (LSPs) for guaranteed QoS flows and a stand-by path should the primary path become unavailable. When determining a primary and alternate path it is crucial to find two paths with minimum interference, in other words, two paths that share the least number of links/nodes. This will result in a greater likelihood that, in the event of a network hardware failure, the alternate route is not affected by that failure. This type of minimum interference routing is based on a single flow request.

Another possible benefit that may be gained from a network monitoring entity is a more efficient use of network resources. During the research for this project, an article published in the IEEE magazine titled, “MPLS Traffic Engineering Using Enhanced Minimum Interference Routing: An Approach Based On Lexicographic Max-Flow” (L-MIRA), by Koushik Kar, Murali Kodialam, and T. V. Laksham was discovered which proposes a new routing algorithm. This routing algorithm also looks at minimum interference issues, but from a larger perspective – it looks at possible interference between the current flow request and all future flow requests. Current routing algorithms are based on the number of hops from source to destination and the bandwidth available on the paths to be used. The L-MIRA algorithm takes into account all ingress and egress routers in the network when making routing decisions. When a flow request arrives at a router the network monitor determines the path which will least interfere with future requests from all ingress/egress routers in the network yet will still satisfy the requirements of the flow request, i.e. bandwidth and delay requirements. This algorithm can be used to determine an efficient primary and alternate path for all flow requests.

This algorithm has been proven to make much better use of network assets (network utilization) than existing algorithms. Although this in itself is very desirable it has the added benefit greatly increasing the chances that a flow can be rerouted in the event of a network failure.

The L-MIRA algorithm is not without its drawbacks though. The most significant drawback of L-MIRA is that it is computationally expensive, which means there may be delays in servicing flow requests. One possible way to reduce the service delay for flow requests would be to only use L-MIRA to determine routes for guaranteed QoS flows. All best efforts flows could be routed according to the old model. Although this best effort traffic might not be routed in the most efficient way it would lighten the load of the network monitor and could always be preempted for higher priority traffic if the need arose. Although L-MIRA can be used to determine a primary and alternate path for an individual flow request, the computational cost involved may eliminate it as the preferred minimum interference routing algorithm, instead a heuristic may be used which is much quicker and still produces satisfactory results.

III. FAILURE DETECTION

A. FAILURE DETECTION STRATEGY

The first consideration when designing a fault tolerant network protocol is a failure detection mechanism. The SAAM auto-configuration cycle will be utilized to detect failures. There are two types of failures that this thesis focuses on, link failures and routers failures. A link failure occurs when an interface fails or the link between two interfaces is broken. A router failure, also referred to as a node failure, is the failure of router hardware or software. A router failure is much more catastrophic than a link failure because in effect it results in link failures for all attached interfaces.

The failure detection method proposed by this thesis does not allow a router to determine whether a detected failure is a link failure or a node failure. If a link failure occurs both routers affected by the failure will send Interface Silent messages to the SAAM server, therefore allowing the SAAM server to determine that a link failure has occurred. The SAAM server can then deduce that a link failure has occurred. If a node failure occurs the SAAM server will receive multiple Interface Silent messages from routers that are attached to the failed node but no Interface Silent messages from the failed router, which will allow the SAAM server to deduce a node failure has occurred.

Each interface has a link state monitor. During each SAAM auto-configuration cycle an interface should receive, at a minimum, one inbound SAAM control message. If no traffic is received at an interface between auto configuration cycles it can be assumed that there is some type of failure causing packets that are supposed to come from the neighboring interface to be dropped.

B. SAAM AUTOCONFIGURATION CYCLE

There are three types of messages that are passed during a SAAM auto-configuration cycle: downward configuration messages (DCM), upward configuration messages (UCM), and parent notifications (PN). The details of these messages are described in chapter III of Hassan Akkoc's thesis, titled "SAAM Signaling Channel Configuration Protocol Design". The content of these messages is not important to the

failure detection mechanism, only the fact that each interface will receive at least one of these messages per auto-configuration cycle.

Each SAAM auto-configuration cycle begins with the SAAM server broadcasting a DCM to all directly connected routers. Those routers return a PN to the node from which they have received the first DCM in the current cycle. The routers then broadcast a DCM to their remaining neighbors. This failure detection method incurs less communication overhead than the existing CISCO failure detection methods, which passes Hello messages between each pair of interfaces. A sufficiently rapid detection time should be achieved since the SAAM auto-configuration cycle is planned to occur at 300-550ms intervals.

1. Pseudo-code for proposed SAAM failure detection scheme

Function `GenerateInterfaceStatusAdvertisement()` is called at each router interface during every configuration cycle. Only relevant code is included in the pseudo-code.

// Trigger event: LSA generation for each auto-configuration cycle.

GenerateInterfaceStatusAdvertisement()

if (`HasTrafficBeenReceived()` = false)

then **SwitchToBackupPaths(thisInterface);**

// end of function GenerateInterfaceStatusAdvertisement

HasTrafficBeenReceived()

if (`lastCyclePacketCount` = `inboundPacketCount`)

then **return** false;

else **return** true;

`lastCyclePacketCount` ← `inboundPacketCount`;

// end of function HasTrafficBeenReceived()

SwitchToBackupPaths(interface)

for each FlowRoutingTableEntry

do if (NextHop & interface are connected & FRTE Type = Primary)

```
then DemoteFlowRoutingTableEntry( FRTE );// Switch to alternate
// paths all FRTE's
```

// that have interface

```
// as the next hop.
```

```
// end of function SwitchToBackupPaths
```

2. Sample SAAM auto-configuration cycle

The following figures illustrate a single SAAM auto-configuration cycle and how each interface receives at least one control message during the cycle.

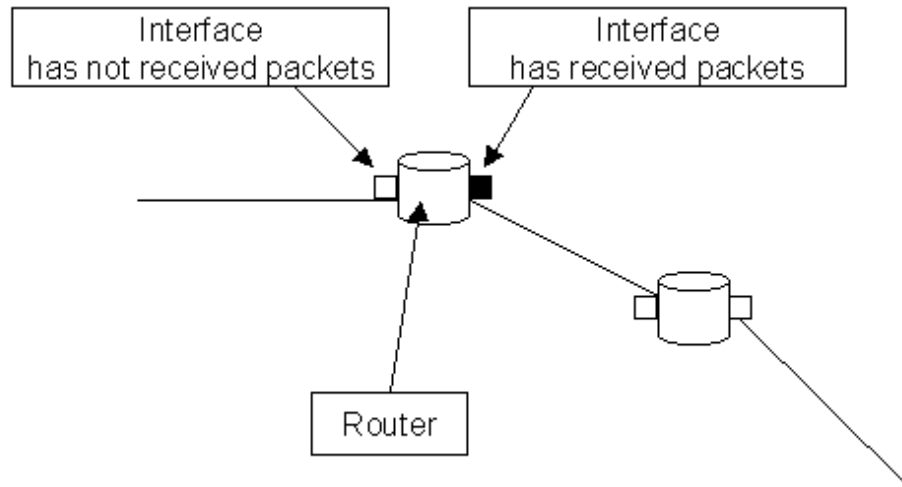


Figure 3.1 Auto-configuration Legend.

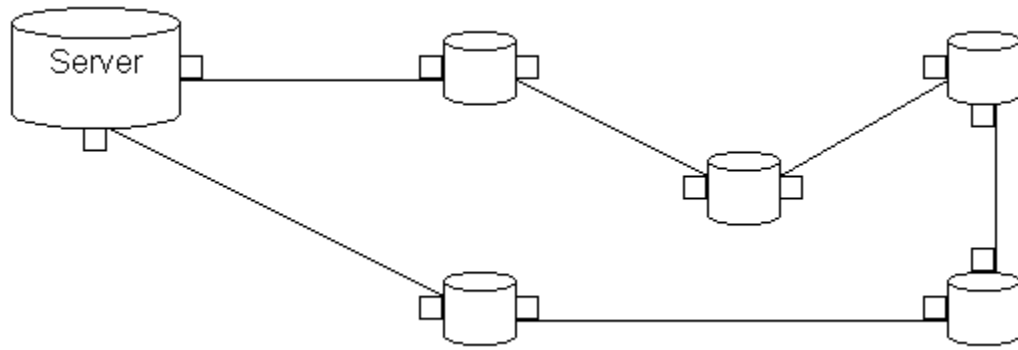


Figure 3.2 Auto-configuration Phase 0.

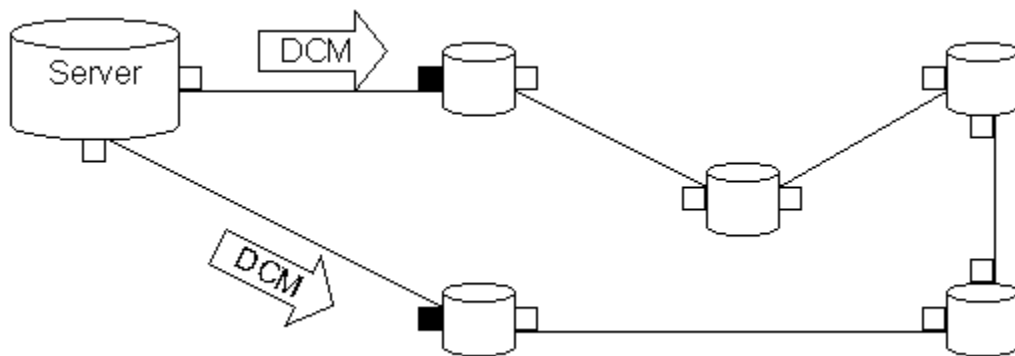


Figure 3.3 Auto-configuration Phase 1.

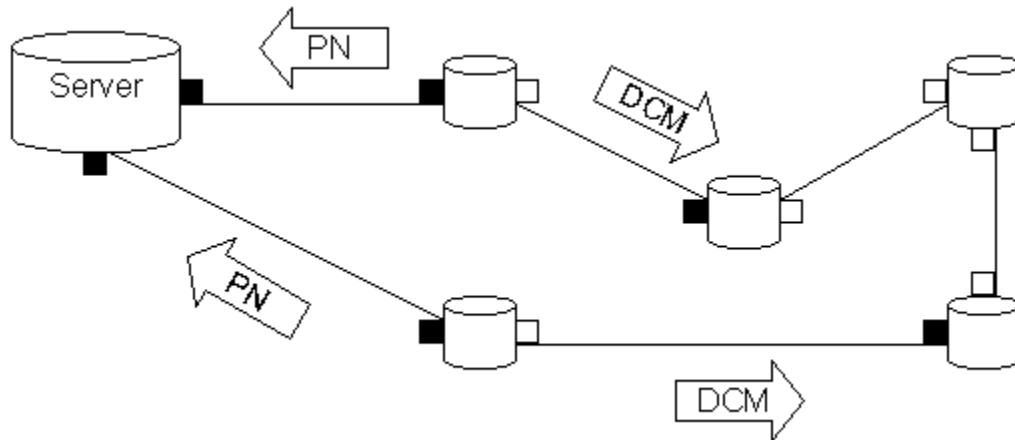


Figure 3.4 Auto-configuration Phase 2.

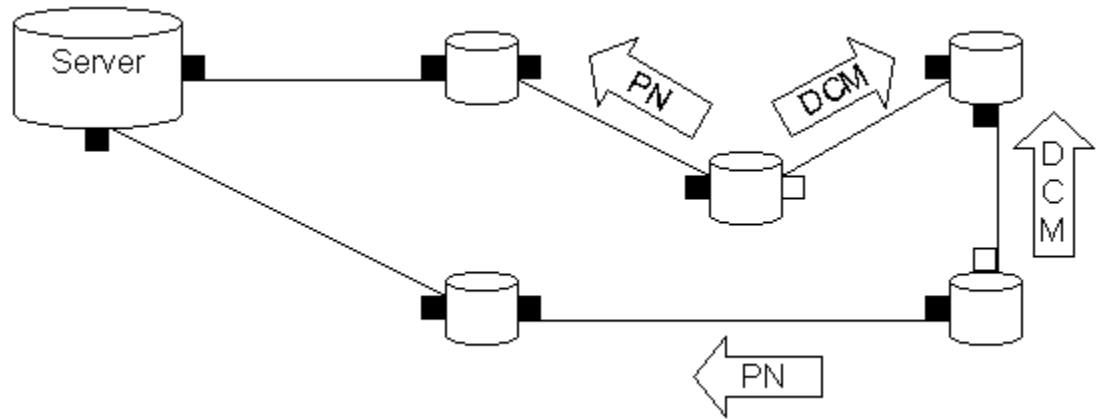


Figure 3.5 Auto-configuration Phase 3.

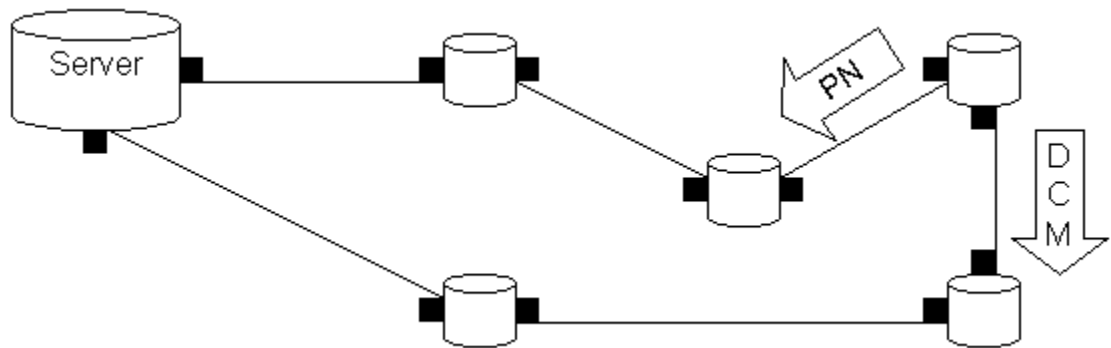


Figure 3.6 Auto-configuration Phase 4.

C. INTERFACE SILENT MESSAGE

Since a local restoration method has been proposed, rerouting traffic can occur without informing the SAAM server that a failure may have occurred. An “Interface Silent” message has been designed though that is sent to the SAAM server to indicate that a router has not received traffic from a particular interface, which indicates a possible failure. The “Interface Silent” message indicates that a router believes that some type of failure has occurred but it is still listening to the interface in question, in the event a quick recovery occurs. At some point, after an interface failure is confirmed, it will be necessary to remove that interface, and all paths that traverse it, from the SAAM network topology. The removal of interfaces from the SAAM network is not formalized by this thesis and is left for further study.

SAAM Link State Advertisement Message Format

Interface Silent Message

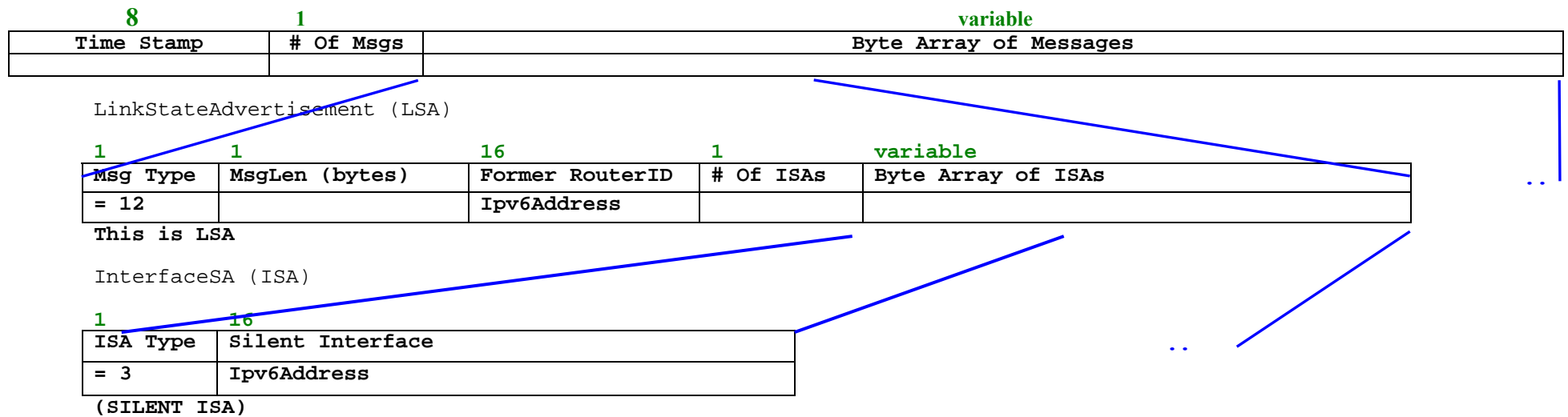


Figure 3.7 Interface Silent Message Format

D. FAILURE DETECTION LIMITATION

The proposed failure detection method has one notable limitation. Specifically, the upstream router is not capable of determining whether a failure is a link failure or a node failure. This is because the failure detection method relies only on detecting that no traffic is being received by a particular interface and makes no attempt to analyze the particular elements that may be causing the failure. Any method that would attempt to differentiate between a link and a node failure would introduce additional overhead, which has been deemed not necessary at this point.

E. CLASSES MODIFIED TO SUPPORT FAILURE DETECTION

1. Interface

a) *public boolean trafficReceived()*

This method is introduced to provide a way to query an interface whether it has received any packets since it was queried last. It is reset each time it is queried.

2. LinkStateMonitor

a) *private synchronized void generateInterfaceSA()*

This method was modified so that each time it is called, which is once per auto-configuration cycle, it in turn calls *Interface.trafficReceived()*. If no traffic has been received an Interface Silent message is generated and sent to the SAAM server.

3. InterfaceSA

a) *public InterfaceSA(IPv6Address ipNum, int messageIndex)*

This is the constructor for the new Interface Silent message. The *messageIndex* is not a necessary field except that it provides a unique constructor for the new message. It is not used in the construction of the actual message. The Interface Silent message is simply the IPv6 address of the interface on which a router is no longer receiving packets.

4. BasePIB

a) *public void processLSA (LinkStateAdvertisement LSA)*

This method modified so that Interface Silent messages are recognized when received. Currently there is no action taken upon the receipt of Interface Silent messages.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. REROUTING

A. REROUTING STRATEGY

A proactive rerouting strategy with local restoration has been chosen. This solution will only provide rerouting services to Integrated Service (IS) flows. An Integrated Service flow is a flow that is introduced into the SAAM based on individual flow requirements. If there are sufficient network resources to meet an IS flow's requirements, i.e. bandwidth, delay, then that flow is accepted and the network guarantees to maintain those parameters which the flow has requested for the duration of the flow.

A proactive rerouting strategy is far superior to a reactive strategy. A reactive strategy is one where alternate path determination is not made until after a failure is detected. This wastes precious time and results in dropped packets. Instead, the SAAM server will be able to establish alternate paths for IS flows before a failure occurs, and then rapidly switch the affected flows to the alternate paths.

1. Alternate Path Development and Deployment

Since the SAAM server maintains a record of all loop-free paths up to a particular hop count in the Path Information Base there is no need to go through the process of path discovery during the alternate path determination process. The alternate path tree is developed by starting at the source node and querying the PIB to determine if there is a path from the source node to the destination node, which avoids the next node in the primary path. We must attempt to avoid the next node because the failure detection method can't distinguish between link and node failures. If there are no paths that avoid the next node check for paths that avoid the bad link and use this as an acceptable second choice. If an alternate path is found send a flow routing table update to the routers in the alternate path. Repeat the alternate path determination process for each node in the primary path, the resulting alternate paths will form a tree rooted at the destination node.

a. Pseudo-code for Alternate Path Development and Deployment

// A Path contains a sequence of nodes and interfaces.


```

then    foundPath  $\leftarrow$  p;

        break;

else if ( p.SourceNode = sourceNode and

          p.DestinationNode = destinationNode and

            previousNode  $\notin$  p and nextInterface  $\notin$  p )

    then    foundPath  $\leftarrow$  p;      // Continue to try and find a
                                     // path which avoids the next
                                     // node.

return foundPath;

// end of FindAlternatePath

```

2. Storing Primary and Alternate Routes

A mechanism must be devised for storing primary and alternate path information at each router. The previous flow routing table was implemented using the Java Hashtable class. In that implementation the path ID is used as the Hashtable index value. The object stored in the flow routing table (Hashtable) is a flow routing table entry, which contains path ID and next hop.

It is desirable to add a “goodness” field to the flow routing table entry, which indicates whether a path is a primary or alternate path. It is also desirable to reference to flow routing table entry objects to a single index value. Hashtable cannot accomplish this because only one object per index value is allowed. Research was done and the JGL class Hashmap was found which has the desired functionality. JGL is a Java Library provided free of cost by the company ObjectSpace. The Universal Resource Locator (URL) of the ObjectSpace website is: <http://www.objectspace.com/>. The JGL library can be download from the ObjectSpace website. This implementation initially stores primary routes with a goodness value of 2 and alternate routes with a goodness value of 1. The routing algorithm will always choose the next hop with the largest goodness value.

Path ID	Next Hop (IPv6)
2	99.99.99.99.0.0.0.0.1.0.0.0.0.0.1
17	99.99.99.99.0.0.0.0.3.0.0.0.0.0.1
49	99.99.99.99.0.0.0.0.3.0.0.0.0.0.1
61	99.99.99.99.0.0.0.0.4.0.0.0.0.0.2

Table 4.1 Example of Old Flow Routing Table.

Path ID	Next Hop (IPv6)	Goodness
2	99.99.99.99.0.0.0.0.1.0.0.0.0.0.1	2
17	99.99.99.99.0.0.0.0.3.0.0.0.0.0.1	2
17	99.99.99.99.0.0.0.0.5.0.0.0.0.0.5	1
49	99.99.99.99.0.0.0.0.3.0.0.0.0.0.1	2
61	99.99.99.99.0.0.0.0.4.0.0.0.0.0.2	2
61	99.99.99.99.0.0.0.0.8.0.0.0.0.0.1	1

Table 4.2 Example of New Flow Routing Table.

3. Local restoration

The primary difference between centralized restoration and local restoration is that the latter provides each router with enough information that it can take appropriate actions if a failure is detected. Since the routers are integral to the failure detection method this seems to be a very appropriate course of action. The primary benefit of local restoration is extremely rapid recovery times.

Local restoration is accomplished by three tasks performed sequentially: detecting a failure, determining which paths are affected by that failure, and then switching the affected path to the alternate next hop. The last task is accomplished by changing the goodness value of the primary path from 2 to 0. The routing algorithm will always choose the next hop with the best (i.e., largest) goodness value. If an alternate path

exists (with goodness == 1) it now becomes the best choice ($1 > 0$) for the given path id. If an alternate did no exist then the best next hop remains the same. See Figure 4.3 for an example of a link failure involving next hop == 99.99.99.99.0.0.0.0.3.0.0.0.0.0.1.

Path ID	Next Hop (IPv6)	Goodness
2	99.99.99.99.0.0.0.0.1.0.0.0.0.0.1	2
17	99.99.99.99.0.0.0.0.3.0.0.0.0.0.1	0
17	99.99.99.99.0.0.0.0.5.0.0.0.0.0.5	1
49	99.99.99.99.0.0.0.0.3.0.0.0.0.0.1	0
61	99.99.99.99.0.0.0.0.4.0.0.0.0.0.2	2
61	99.99.99.99.0.0.0.0.8.0.0.0.0.0.1	1

Table 4.3 Flow Routing Table after Switching to Backup Path(s).

B. CLASSES MODIFIED TO SUPPORT REROUTING

1. FlowRoutingTable

The major change to the **Flow Routing Table** class is that it now extends the **JGL Hashmap** class.

a) public synchronized void add (FlowRoutingTableEntry entry)

This method is overridden so that no to flow routing table entries contain the same next hop and goodness value.

b) public Object get(Object o)

This method is overridden since the Hashmap may contain more than one object per index value. This method returns the object with the best (highest) goodness value.

c) public void silentInterface(Interface badInterface)

This method is introduced to switch a path to its backup route, if one exists. It attempts to switch all paths that contain **badInterface** as their next hop.

2. **FlowRoutingTableEntry**

a) ***public int getGoodness()***

The primary change to the flow routing table entry is the addition of a goodness field and a method to access the value in that field.

3. **BasePIB**

a) ***private Path findAltPath(Integer sourceNode, Integer destinationNode, Integer previousNode, Integer nextNode, IPv6Address deadInterface)***

This method determines whether there is a path that exists which avoids the previous router and the next router. If the next router cannot be avoided it determines if the next interface can be avoided as the second best solution.

b) ***protected void createAlternateTree(Path primaryPath)***

This method iterates over the nodes traversed by a primary path and creates an tree of alternate paths, rooted at the destination node. The alternate tree is set up by this method by sending new flow routing table entries to the appropriate routers.

c) ***protected int admissionControl_IS(FlowRequest flowRequest)***

This method is where ***createAlternateTree(Path primaryPath)*** is called from, ensuring that the additional overhead of alternate path development is only incurred if an IS flow is traversing a path. The ***primaryPath*** passed to ***createAlternateTree(Path primaryPath)*** is the primary path of the IS flow.

IV. TESTING

A. ALTERNATE PATH(S)/TREE TESTING

Numerous network topologies were used to test the failure detection and alternate path(s)/tree development algorithm. Each topology diagram is accompanied by a description of the expected results and the resulting flow routing tables from the topology test. A flow routing table with more than one entry for a path ID indicates that an alternate path has been established. An initial goodness value of 2 denotes a primary path, whereas an initial goodness value of 1 denotes an alternate path. Path ID #2, seen in all the tests, is used for SAAM control traffic.

1. No alternate paths topology

The topology in Figure 5.1 tests to ensure that no alternate paths are found for flows from router “A” to router “C”.

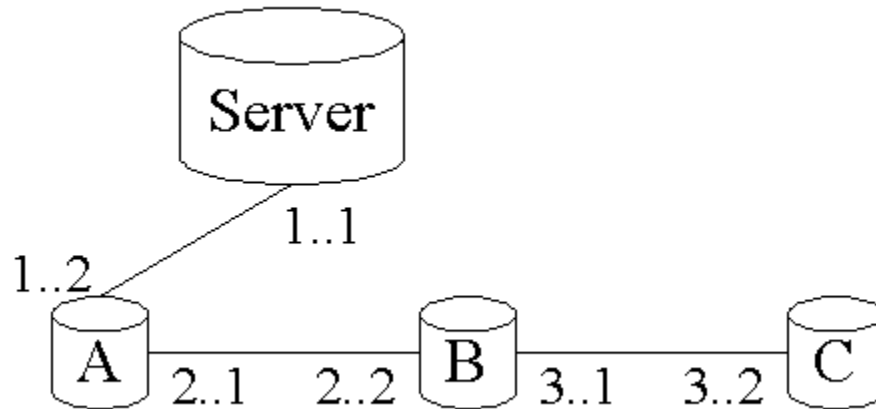


Figure 5.1 No alternate path topology.

Router A [Demo-EmulationPort: 9004][SaamPort: 9003] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.1.0.0.0.0.0.0.0.0.0.1	2	
76	1	99.99.99.99.2.0.0.0.0.0.0.0.0.0.2	2	

Router B [Demo-EmulationPort: 9006][SaamPort: 9005] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.2.0.0.0.0.0.0.0.0.0.1	2	
76	1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.2	2	

Router C [Demo-EmulationPort: 9008][SaamPort: 9007] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.3.0.0.0.0.0.0.0.0.0.1	2	

Table 5.1 No Alternate Path Routing Tables.

2. Avoid next interface on last hop topology

The topology in Figure 5.2 uses a flow request from router “A” to router “C” to ensure that when the next hop cannot be avoided an attempt is made to avoid the next interface. This also handles the alternate path from the node just prior to the destination.

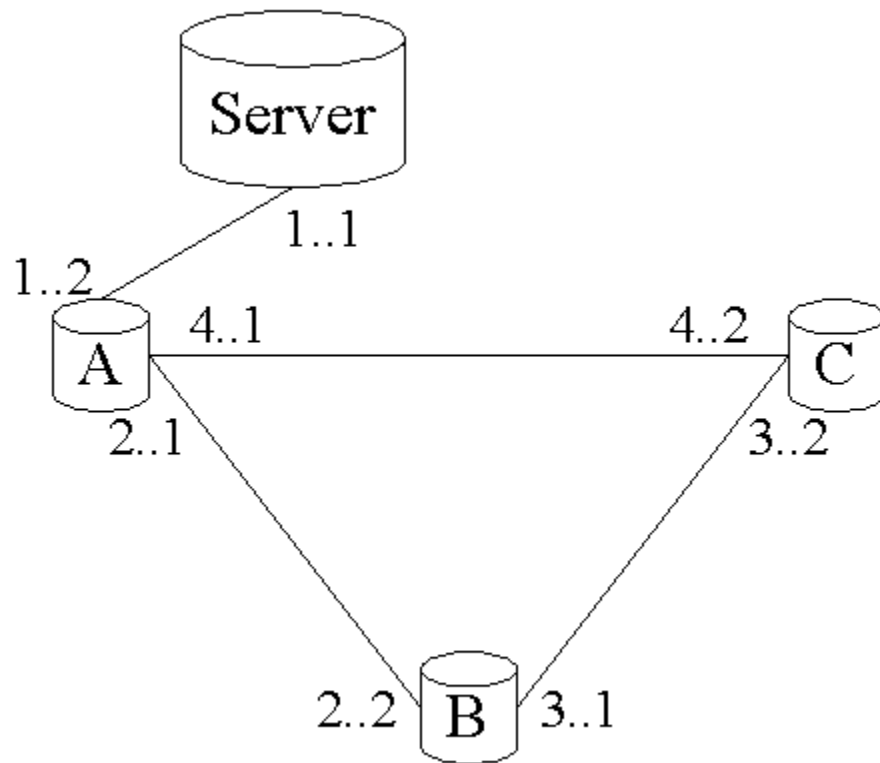


Figure 5.2 Avoid next interface topology.

Router A [Demo-EmulationPort: 9004][SaamPort: 9003] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1	2	
78	2	99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2	2	
78	1	99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2	1	

Router B [Demo-EmulationPort: 9006][SaamPort: 9005] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1	2	
78	1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2	1	

Router C [Demo-EmulationPort: 9008][SaamPort: 9007] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	1	99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1	2	

Table 5.2 Avoid Next Interface Routing Tables.

3. Avoid infinite loop topology

The topology in Figure 5.3 tests to ensure that one alternate path is found from router “A” to router “C” and that no infinite loop is established between router “A” and router “B”.

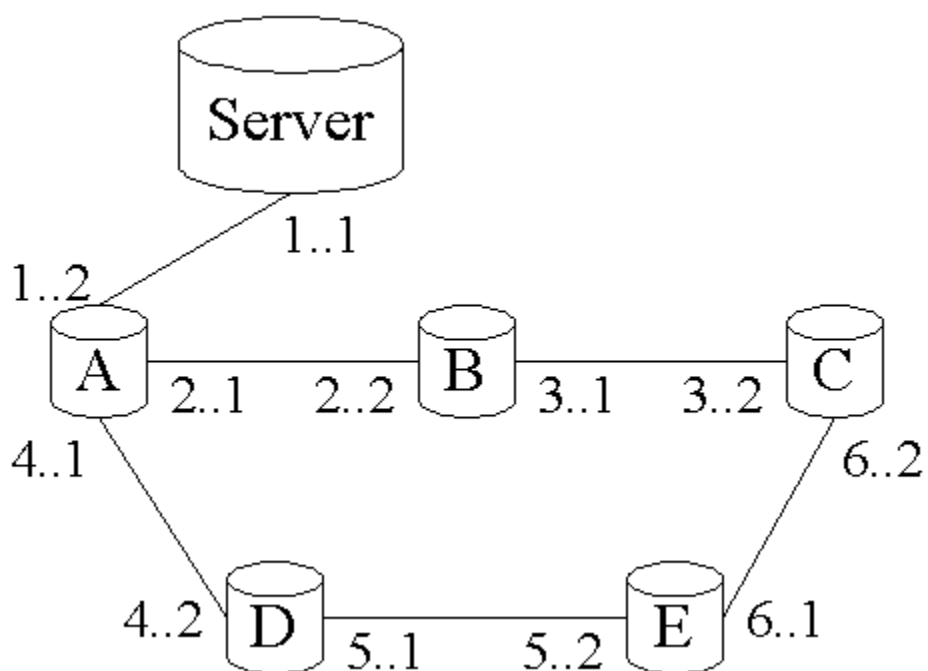


Figure 5.3 No infinite loops topology.

Router A [Demo-EmulationPort: 9004][SaamPort: 9003] [Currently displaying: Flow Routing Table]

File Protocol Stack Routing Tables Open Channels Active Ports SLSTable Flow Tables Application Agents

Path ID	Interface ...	Next Hop IPv6 Address	Goodness
2	0	99.99.99.99.1.0.0.0.0.0.0.0.0.0.1	2
90	1	99.99.99.99.2.0.0.0.0.0.0.0.0.0.2	2
90	2	99.99.99.99.4.0.0.0.0.0.0.0.0.0.2	1

Router B [Demo-EmulationPort: 9006][SaamPort: 9005] [Currently displaying: Flow Routing Table]

File Protocol Stack Routing Tables Open Channels Active Ports SLSTable Flow Tables Application Agents

Path ID	Interface ...	Next Hop IPv6 Address	Goodness
2	0	99.99.99.99.2.0.0.0.0.0.0.0.0.0.1	2
90	1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.2	2

Router C [Demo-EmulationPort: 9008][SaamPort: 9007] [Currently displaying: Flow Routing Table]

File Protocol Stack Routing Tables Open Channels Active Ports SLSTable Flow Tables Application Agents

Path ID	Interface ...	Next Hop IPv6 Address	Goodness
2	0	99.99.99.99.3.0.0.0.0.0.0.0.0.0.1	2

Router D [Demo-EmulationPort: 9010][SaamPort: 9009] [Currently displaying: Flow Routing Table]

File Protocol Stack Routing Tables Open Channels Active Ports SLSTable Flow Tables Application Agents

Path ID	Interface ...	Next Hop IPv6 Address	Goodness
2	0	99.99.99.99.4.0.0.0.0.0.0.0.0.0.1	2
90	1	99.99.99.99.5.0.0.0.0.0.0.0.0.0.2	1

Router E [Demo-EmulationPort: 9012][SaamPort: 9011] [Currently displaying: Flow Routing Table]

File Protocol Stack Routing Tables Open Channels Active Ports SLSTable Flow Tables Application Agents

Path ID	Interface ...	Next Hop IPv6 Address	Goodness
2	0	99.99.99.99.5.0.0.0.0.0.0.0.0.0.1	2
90	1	99.99.99.99.6.0.0.0.0.0.0.0.0.0.2	1

Table 5.3 No Infinite Loops Flow Routing Tables.

4. Avoid next node topology

The topology in Figure 5.4 tests a flow request from router “A” to router “C” to ensure that the next node is avoided when possible.

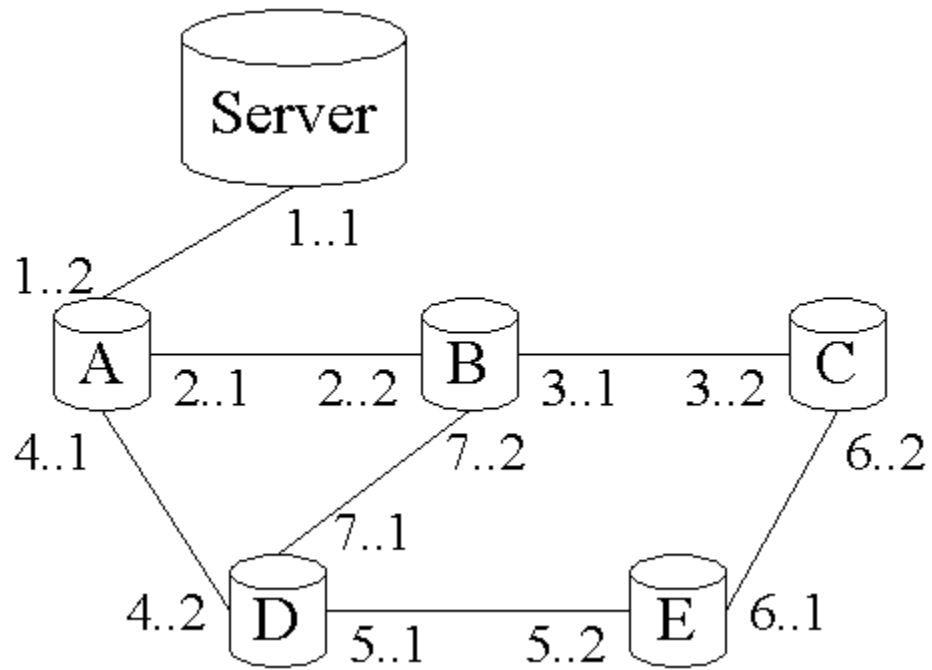


Figure 5.4 Avoid next node topology.

Router A [Demo-EmulationPort: 9004][SaamPort: 9003] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.1.0.0.0.0.0.0.0.0.0.1	2	
76	1	99.99.99.99.2.0.0.0.0.0.0.0.0.0.2	2	
76	2	99.99.99.99.4.0.0.0.0.0.0.0.0.0.2	1	

Router B [Demo-EmulationPort: 9006][SaamPort: 9005] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.2.0.0.0.0.0.0.0.0.0.1	2	
76	1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.2	2	

Router C [Demo-EmulationPort: 9008][SaamPort: 9007] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.3.0.0.0.0.0.0.0.0.0.1	2	

Router D [Demo-EmulationPort: 9010][SaamPort: 9009] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.4.0.0.0.0.0.0.0.0.0.1	2	
76	1	99.99.99.99.5.0.0.0.0.0.0.0.0.0.2	1	

Router E [Demo-EmulationPort: 9012][SaamPort: 9011] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	1	99.99.99.99.6.0.0.0.0.0.0.0.0.0.2	2	
76	1	99.99.99.99.6.0.0.0.0.0.0.0.0.0.2	1	

Table 5.4 Avoid Next Node Flow Routing Tables.

5. Avoid next interface topology

The topology in Figure 5.5 tests a router “A” to router “C” flow request to ensure that avoid next interface works properly.

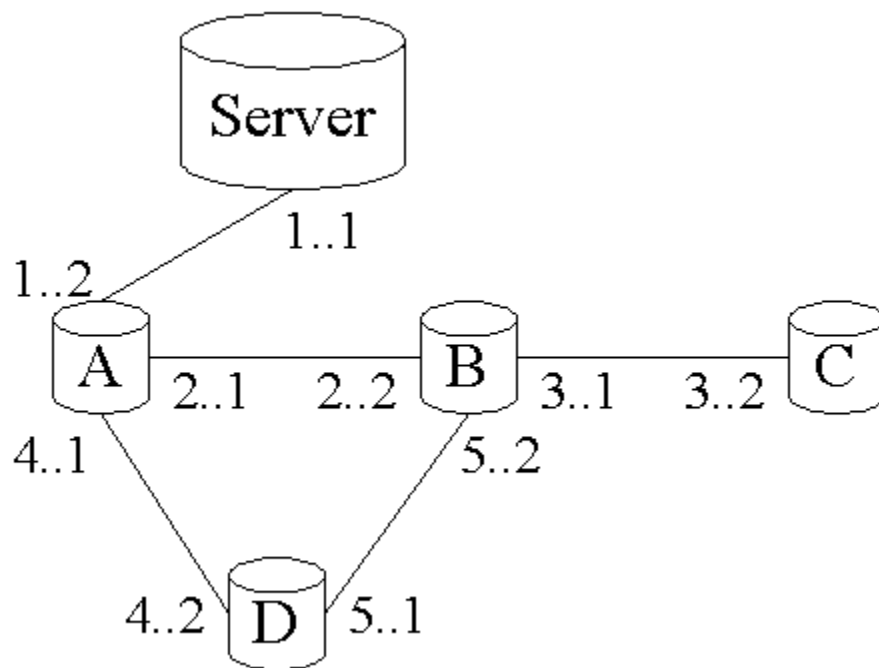


Figure 5.5 Avoid next interface topology.

Router A [Demo-EmulationPort: 9004][SaamPort: 9003] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.1.0.0.0.0.0.0.0.0.0.1	2	
94	1	99.99.99.99.2.0.0.0.0.0.0.0.0.0.2	2	
94	2	99.99.99.99.4.0.0.0.0.0.0.0.0.0.2	1	

Router B [Demo-EmulationPort: 9006][SaamPort: 9005] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.2.0.0.0.0.0.0.0.0.0.1	2	
94	1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.2	2	
94	1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.2	1	

Router C [Demo-EmulationPort: 9008][SaamPort: 9007] [Currently displaying: Flow Routing Table]				
File	Protocol Stack	Routing Tables	Open Channels	Active Ports
SLSTable	Flow Tables	Application Agents		
Path ID	Interface ...	Next Hop IPv6 Address	Goodness	
2	0	99.99.99.99.3.0.0.0.0.0.0.0.0.0.1	2	

Table 5.5 Avoid Next Interface Flow Routing Tables.

B. TRAFFIC REROUTING TESTING

Local restoration is accomplished by three tasks performed sequentially: detecting a failure, determining which paths are affected by that failure, and then switching the

affected path to the alternate next hop. The last task is accomplished by changing the goodness value of the primary path from 2 to 0. The routing algorithm will always choose the next hop with the best (i.e., largest) goodness value. If an alternate path exists (with goodness == 1) it now becomes the best choice ($1 > 0$) for the given path id. If an alternate did no exist then the best next hop remains the same. See Figure 4.3 for an example of a link failure involving next hop == 99.99.99.99.0.0.0.0.3.0.0.0.0.0.1.

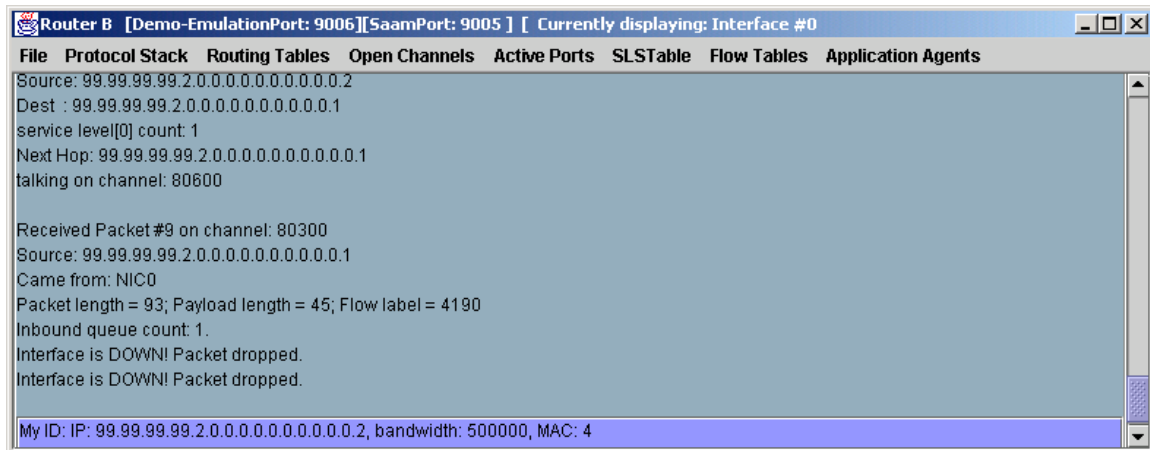


Table 5.6 Router “B” Interface Status

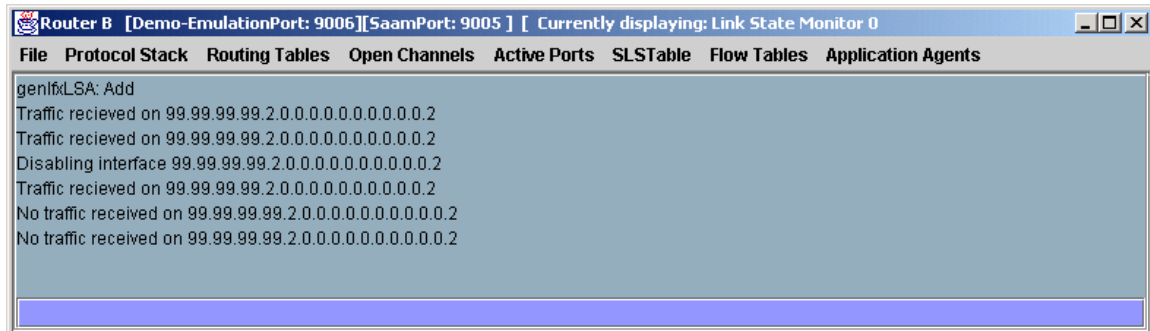


Table 5.7 Router “B” Link State Monitor

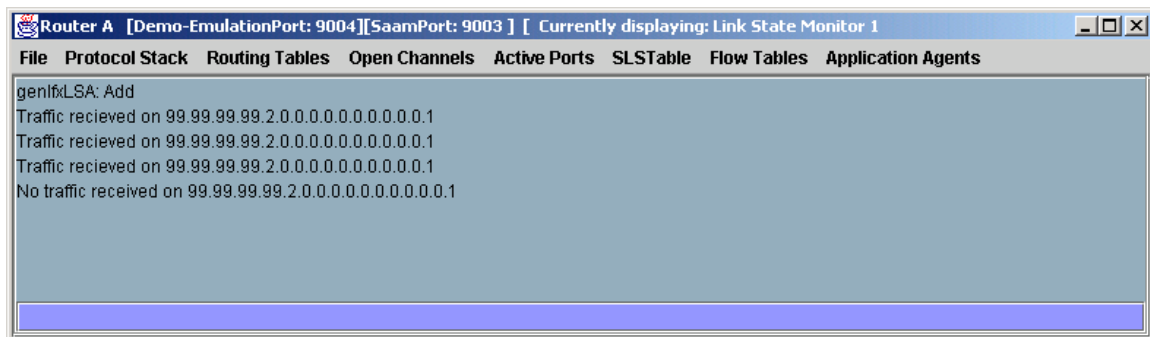


Table 5.8 Router “A” Link State Monitor

Path ID	Interface ...	Next Hop IPv6 Address	Goodness
2	0	99.99.99.99.1.0.0.0.0.0.0.0.0.0.1	2
94	1	99.99.99.99.2.0.0.0.0.0.0.0.0.0.2	0
94	2	99.99.99.99.4.0.0.0.0.0.0.0.0.0.2	1

Table 5.9 Router “A” Flow Routing Table

installing...

Monitoring port: 6011

Got a packet from:
99.99.99.99.2.0.0.0.0.0.0.0.0.0.1 pkt#: 1 payload len = 45
sequence #: 1

Got a packet from:
99.99.99.99.2.0.0.0.0.0.0.0.0.0.1 pkt#: 2 payload len = 45
sequence #: 2

Got a packet from:
99.99.99.99.2.0.0.0.0.0.0.0.0.0.1 pkt#: 3 payload len = 45
sequence #: 3

Got a packet from:
99.99.99.99.2.0.0.0.0.0.0.0.0.0.1 pkt#: 4 payload len = 45
sequence #: 4

Got a packet from:
99.99.99.99.2.0.0.0.0.0.0.0.0.0.1 pkt#: 5 payload len = 45
sequence #: 10

Got a packet from:
99.99.99.99.2.0.0.0.0.0.0.0.0.0.1 pkt#: 6 payload len = 45
sequence #: 11

Got a packet from:
99.99.99.99.2.0.0.0.0.0.0.0.0.0.1 pkt#: 7 payload len = 45
sequence #: 12

I'm on 192.168.0.1 Monitoring Port 6011

Table 5.10 Router “C” Flow Sink

V. CONCLUSIONS AND RECOMMENDATIONS

A. SYNOPSIS AND CONCLUSION

This thesis determined the most suitable traffic rerouting scheme for the Server and Agent Based Active Network Management project. Most research on network fault tolerance has followed a strict adherence to either a local or central restoration algorithm. The hybrid fault tolerance schema designed for SAAM allows a SAAM network to benefit from the strengths of each methodology. The SAAM Server's total knowledge of the region's architecture allows the optimization of alternative route development. The ability of the individual routers to use this alternate path information at a local level allows for fast switching to backup paths and minimal loss of packets.

B. FLOW ROUTING TABLE REDESIGN

The previous Flow Routing Table was implemented with the native Java Hashtable. Hashtable is the logical entity to use since it performs very fast lookups on index values – a critical feature in a router. To allow local restoration a Path must have more than one option for Next Hop. Since Flow Routing Table entries are indexed by Path ID and Java's Hashtable can contain no more than one Object per index a work-around had to be found. JGL's Hashmap class fills the requirement exactly. It performs all the same functions as Hashtable and allows more than one Object per index value. Since more than one Flow Routing Table Entry is stored per index value a method of determining which entry is the best must be available, hence the addition of the Goodness field.

C. FAILURE DETECTION

Keeping in mind that failures are becoming scarcer, the task was to develop a method of detecting network failures with minimal overhead. The conventional wisdom for monitoring the health of a network has been to send "heartbeat" packets between routers and interfaces. Since the SAAM auto-configuration cycle occurs continuously, and at a set rate, this traffic can be used in the place of heartbeat packets. By queuing off the configuration cycles, which provides automatic synchronization, and monitoring each interface for inbound packets received, an extremely low-overhead failure detection

method was designed. The frequency of the auto-configuration messages is configurable, but is predicted to be in the 300-500ms range. Detecting a network failure in less than 1000ms was determined to be very acceptable indeed.

D. TEST METHODOLOGY

The test methodology for this thesis is not complex. The two most important factors infinite loops, minimal interference were addressed by just a few topologies. The topologies used for testing are simple but are indicative of current network connectivity patterns.

E. AREAS FOR FURTHER INVESTIGATION AND STUDY

Reiterating the premise that network failures are scarce and overall network performance should not see significant negative impact by the implementation of a fault tolerance schema, the weaknesses of this thesis are highlighted.

Alternate path development does not consider flow requirements. The alternate path development algorithm simply takes the first shortest path found which has minimal interference with the primary path. This is because in SAAM flows are switched based on Path ID. Although it is possible to switch based on Flow IDs this would result in routing tables that would be astronomically large and unmanageable.

Alternate path bandwidth sharing. The initial design of SAAM makes this a simple problem since it divides the bandwidth up into 10% control traffic, 30% Integrated Service, 30% Differentiated Service, and 30% Best Effort. Since Integrated Service flows have priority and they take up only 30% of the bandwidth they can simply preempt lower class traffic on an alternate path, if necessary. This preemption will be taken care of by the priority scheduler. Paulo Silva's thesis introduces the concept of inter-service borrowing, which complicates an otherwise simple solution. With inter-service borrowing it will be possible for Integrated Services to consume more than 50% of the bandwidth and therefore lower class traffic preemption may not solve the alternate path bandwidth sharing problem. One possible solution to this problem would be to assign two logical Path IDs, a primary path and a borrowing path, to the same physical path and to only allow up to 30% of IS traffic on the primary path. In the event of a failure only the traffic on the primary path would be switched to the alternate path. One obvious

drawback to this solution is that it would double the size of the routing tables. This is left to further investigation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. FLOW ROUTING TABLE CODE CHANGES

```
package saam.agent.router;
import com.objectspace.jgl.HashMap;

public class FlowRoutingTable extends HashMap
implements TableResidentAgent, MessageProcessor, FlowRoutingTableResidentAgent
{
    //tests if FlowRoutingTable contains an entry for a Path
    public boolean hasEntry(int pathID)
    {
        if (super.get(new Integer(pathID)) != null)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    //Returns the FlowRoutingTableEntry with the highest goodness rating
    public Object get(Object o)
    {
        int bestGoodness = 0;
        Object bestChoice = null;
        Object tempChoice = null;
        FlowRoutingTableEntry frte;
        Enumeration e = values(o);

        while (e.hasMoreElements())
        {
            tempChoice = e.nextElement();
            frte = (FlowRoutingTableEntry) tempChoice;
            if (frte.getGoodness() > bestGoodness)
            {
                bestChoice = tempChoice;
                bestGoodness = frte.getGoodness();
            }
        }
        return bestChoice;
    }
}
```

```

public void silentInterface(Interface badInterface)
{
    //Enumerate entire FRT looking for entries which == badNextHop
    Enumeration e = elements();
    FlowRoutingTableEntry frte = null;

    while (e.hasMoreElements())
    {
        frte = (FlowRoutingTableEntry) e.nextElement();

        if (Interface.isOnSameNetwork(frte.getNextHop(),
            badInterface.getID().getIPv6(), badInterface.getID().getSubnetMask())
            && frte.getGoodness() == FlowRoutingTableEntry.PRIMARY_ROUTE)
        {
            frte.setGoodness(
                FlowRoutingTableEntry.DEMOTE_PRIMARY_ROUTE );
        }
    }
    gui.fillTable(getTable()); //refresh FRT
}

public synchronized void add (FlowRoutingTableEntry entry)
{
    //Ensure there are no entrys with equal goodness fields for any one path
    Enumeration ptr = elements();
    Enumeration e = elements();
    while (ptr.hasMoreElements())
    {
        FlowRoutingTableEntry frte = (FlowRoutingTableEntry) e.nextElement();
        if (frte.getFlowLabel() == entry.getFlowLabel() && frte.getGoodness() ==
            entry.getGoodness())
        {
            remove( ptr ); //This removes a FRTE that is being updated
        }
        ptr.nextElement();
    }

    add((new Integer(entry.getFlowLabel())), entry); //[TW]
    gui.fillTable(getTable());
}

private void remove (FlowRoutingTableEntry entry)
{
    remove(new Integer(entry.getFlowLabel())); //Hashtable's remove method
    gui.fillTable(getTable());
}

```

}APPENDIX B. BASEPIB CODE CHANGES

```
/**
 * Creates the alternate path/tree for IS flows
 * @param primaryPath the primary path for which an alternate tree is being developed
 * @return void
 */
protected void createAlternateTree(Path primaryPath)
{
    Integer srcNode, dstNode, prvNode, nxtNode;
    IPv6Address deadInterface;
    Integer primarySrcNode = primaryPath.getPIIndex().getSource();
    Integer primaryDstNode = primaryPath.getPIIndex().getDestination();
    Path backupPath;

    int nodeSeqSize = primaryPath.getNodeSequence().size();
    int[] nodeSeq = new int[nodeSeqSize];
    int nodeSeqNum = 0;
    Enumeration eNodeSeq = primaryPath.getNodeSequence().elements();
    while (eNodeSeq.hasMoreElements())
    {
        nodeSeq[nodeSeqNum] = ((Integer) eNodeSeq.nextElement()).intValue();
        nodeSeqNum++;
    }

    int interfaceSeqSize = nodeSeqSize - 1;
    IPv6Address[] interfaceSeq = new IPv6Address[interfaceSeqSize];
    int interfaceSeqNum = 0;
    Enumeration eInterfaceSeq = primaryPath.getInterfaceSequence().elements();
    while (eInterfaceSeq.hasMoreElements())
    {
        interfaceSeq[interfaceSeqNum] = ((IPv6Address) eInterfaceSeq.nextElement());
        display.sendText("interface " + interfaceSeqNum +
            interfaceSeq[interfaceSeqNum]);
        interfaceSeqNum++;
    }

    for (int x = nodeSeqSize - 1; x > 0; x--)
    {
        srcNode = new Integer(nodeSeq[x]);
        if (x != nodeSeqSize - 1)
        {
            prvNode = new Integer(nodeSeq[x + 1]);
        }
        else
    }
```

```

    {
        prvNode = null;
    }
    nxtNode = new Integer(nodeSeq[x - 1]);
    dstNode = primaryDstNode;
    deadInterface = interfaceSeq[x - 1];

    backupPath = routingAlgorithm.findAltPath(
        srcNode,
        dstNode,
        prvNode,
        nxtNode,
        deadInterface);

    if (backupPath != null)
    {
        setupPath(backupPath, primaryPath.getPathID().intValue(),
            FlowRoutingTableEntry.BACKUP_ROUTE);
        primaryPath.bBackupCreated = true;
    }
    else
    {
        display.sendText("No backup route found!!");
        display.sendText("dead interface " + deadInterface);
    }
}
} // end of createAlternateTree()

```

```

/**
 * Implementation of the First Shortest Path algorithm for finding alternate paths.
 * @param sourceNode the source router ID
 * @param destinationNode the destination router ID
 * @param previousNode the route ID of the previous router
 * @param nextNode the router ID of the next router
 * @param deadInterface the IPv6 address of the next interface
 * @return the required path or null if no path was found
 */
private Path findAltPath(Integer sourceNode, Integer destinationNode,
    Integer previousNode, Integer nextNode, IPv6Address deadInterface)
{
    testMsg("findAltPath() for rerouting");

    Path checkPath = null;
    Path foundPath = null;

```

```

Hashtable table = new Hashtable();

//labeled compound statement
stop:
{
    for (int i = 1; i < MAX_HOP_COUNT; i++ )
    {
        testMsg("Hop count = " + i);
        table = aPI[sourceNode.intValue()][destinationNode.intValue()][i];
        Enumeration enum = table.elements();

        if (enum.hasMoreElements())
        {
            //Cycle through each of the paths of the current hop count, between
            //source and destination nodes
            while (enum.hasMoreElements())
            {
                Integer currentPathID = (Integer) enum.nextElement();

                // Extract current path information
                checkPath = (Path) htPaths.get(currentPathID);

                if (!(checkPath.containsNode(previousNode)) &&
                    !(checkPath.containsNode(nextNode)))
                {
                    foundPath = checkPath;
                    break stop;
                }
                else if (!(checkPath.containsInterface(deadInterface)) &&
                    !(checkPath.containsNode(previousNode)))
                {
                    if (foundPath == null) //hold the first one found
                    {
                        foundPath = checkPath;
                    }
                }
            }
        } //End of while-loop
    } //End of if structure
} //End of for-loop
} //End of labeled stop structure

return foundPath;

} //end of findAltPath()

```


THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Kodialam, M., Lakshman, T.V., “Dynamic Routing of Locally Restorable Bandwidth Guaranteed Tunnels using Aggregated Link Usage Information”, INFOCOM 2001, IEEE Proceedings, Volume: 1, 2001
2. Kar, K., Kodialam, M., Lakshman, T.V., “Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications”, IEEE Journal on Selected Areas in Communications, Volume: 18 Issue: 12, December 2000
3. Kodialam, M., Lakshman, T.V., “Dynamic Routing of Bandwidth Guaranteed Tunnels With Restoration”, INFOCOM 2000, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE Proceedings, Volume: 2, 2000
4. Kuo Dao-Cheng, John H. Gibson, “Design of a Dynamic Management Capability for the Server and Agent Based Active Network Management (SAAM) System to Support Requests for Guaranteed Quality of Service Traffic Routing and Recovery”, Master’s Thesis September 2000, Computer Science Department, Naval Postgraduate School
5. Hasan Akkoc, “A Pro-Active Routing Protocol for Configuration of Signaling Channels in Server and Agent-Based Active Network Management (SAAM)”, Master’s Thesis September 2000, Computer Science Department, Naval Postgraduate School
6. Efraim Kati, “Fault-Tolerant Approach for Deploying Server Agent-Based Active Network Management (SAAM) Server in Windows NT Environment to Provide Uninterrupted Services to Routers in Case of Server Failure(s)”, Master’s Thesis September 2000, Computer Science Department, Naval Postgraduate School

THIS PAGE INTENTIONALLY LEFT BLANK

DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Marine Corps Representative
Naval Postgraduate School
Monterey, California
debarber@nps.navy.mil
4. Director, Training and Education, MCCDC, Code C46
Quantico, Virginia
webmaster@tecom.usmc.mil
5. Director, Marine Research Center, MCCDC, Code C40RC
Quantico, Virginia
ramkeyce@tecom.usmc.mil
strongka@tecom.usmc.mil
sanftlebenka@tecom.usmc.mil
6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
doranfv@mctssa.usmc.mil
palanaj@mctssa.usmc.mil
7. Dr. Mari W. Maeda
Program Manager
DARPA/ITO
3701 Fairfax Drive
Arlington, VA 22203-1714
mmaeda@darpa.mil
8. Cary Colwell
Naval Postgraduate School
Monterey, California
colwell@cs.nps.navy.mil